Freeform Search

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** TI **Starting with Number** 1

Generate: ○ Hit List ◉ Hit Count ○ Side by Side ○ Image

Search | Clear | Interrupt

Search History

DATE: Tuesday, May 11, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L9	L8 and l2	10	L9
L8	simulat$6 adj10 thread	212	L8
L7	L6 and l1	3	L7
L6	dynamic$6 adj2 thread	73	L6
L5	dynamic$6-allocated thread	0	L5
L4	dynamic$6 allocated thread	0	L4
L3	L2 and l1	12	L3
L2	718/$.ccls.	2894	L2
L1	simulat$6 near5 thread	228	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection

L3: Entry 1 of 12 File: USPT Mar 18, 2003

DOCUMENT-IDENTIFIER: US 6535878 B1
TITLE: Method and system for providing on-line interactivity over a server-client network

Brief Summary Text (27):
Modern operating systems support multi-tasking, which is the ability to run many separate applications at the same time. A single software program can take advantage of multi-tasking by creating multiple concurrent "threads." Each thread simulates a separate application. Thus an HTTP server, for example, can use multiple multiple threads to optimize its performance in responding to concurrent requests. Each request can be processed in a separate thread, and while one request is being processed in the CPU, a second request can be transmitted through the network hardware. If only one thread is used, then although the network hardware is buffered, the processing of the second request can become blocked--because the single thread waits for the network to finish sending.

Current US Cross Reference Classification (3):
718/102

Current US Cross Reference Classification (4):
718/108

L3: Entry 2 of 12 File: USPT Mar 4, 2003

DOCUMENT-IDENTIFIER: US 6529985 B1
TITLE: Selective interception of system calls

Detailed Description Text (27):
User level threads do not utilize operating system resources, or provide bona fide threads of execution. Instead, user level threads simulate multiple threads of execution with a set of localized library functions. Because operating system resources are not employed to create the threads, the operating system has no record of the existence of the threads. Therefore, the simulated threads are unable to interact with the operating system as if they were actual independent threads of execution. Because of this, only one system call 115 per process can be serviced at a time, and no execution flag 131 conflicts can occur.

Current US Cross Reference Classification (1):
718/107

First Hit Fwd Refs

Generate Collection

L3: Entry 9 of 12 File: USPT Nov 10, 1998

DOCUMENT-IDENTIFIER: US 5835763 A
TITLE: Threaded environment for computer systems without native threading support

Brief Summary Text (16):
In accordance with the present invention, an additional layer of tasks or jobs is provided between a conventional user application and a series of conventional, specialized batch jobs on which the user application may schedule work to be done in parallel, that is, in a manner simulating the behavior of asynchronous application threads. This additional layer is known herein as the thread enabling layer and includes a single application queue, called the common or API queue, on which the user application waits and to which all subtasle outputs--or returns--are provided. Wherever data from any of the subtasks is placed on the API queue, the waiting user application wakes up and interrogates the queue to determine which subtask, or application thread, has produced the data and then the application retrieves the data.

Brief Summary Text (19):
In this manner, the thread enabling layer serves as an effectively asynchronous interface between an user application and a series of specialized, target batch jobs whose behavior with regard to the user application has been made to simulate the behavior of a series of application threads.

Detailed Description Text (3):
The modifications to non-threaded computer system 10 to permit the use of actual or simulated threads could be made at the hardware, microcode, operating system or applications level of the hardware software complex which forms the computer system. In accordance with the present invention, however, a threaded environment is created within non-threaded computer system 10 by modification only of the application layer of the hardware software complex.

Detailed Description Text (15):
Referring now to FIG. 2, threads such as thread jobs 22, 25 and 26 are conventional non-threaded computer batch jobs created by API 24 using native batch job facilities. Although such thread jobs may be created one at a time by API when needed, a substantial reduction of overhead may be achieved in accordance with a after aspect of the present invention by using a thread pool with clonable threads. API 24 includes thread manager 38 which, when enabled, creates and maintains thread pool 40. Thread pool 40 comprises one or more conventional non-threaded computer batchjobs setup for use in the present invention to provide the simulation of threads. The primary difference between the threads in thread pool 40 and other non-non-threaded computer batch jobs is that although such batch jobs are typically totally independent of each other, the threads in thread pool 40 are tied together to be made aware of each other. The threads in pool 40 are coded in C or any other convenient language and are main entry points for the program.

Current US Original Classification (1):
718/101

Current US Cross Reference Classification (1):
718/107

First Hit Fwd Refs

Generate Collection

L3: Entry 10 of 12 File: USPT Jan 21, 1997

DOCUMENT-IDENTIFIER: US 5596579 A
TITLE: High performance machine for switched communications in a heterogeneous data processing network gateway

Detailed Description Text (49):
Three workstations running the Distributed Computing Environment (DCE) of the Open Software Foundation (OSF) on a LAN were used in this experimental set-up. One workstation was for clients issuing incoming switched calls, a second was for clients issuing outgoing switched calls and a third for running a server emulating a LAN gateway. Multiple threads within one process were used to simulate multiple clients.

Detailed Description Text (54):
A timer thread 56 (FIG. 4) was used to simulate incoming calls from a switched network to the server. Upon waking up by the system following a sleep time corresponding to interarrival of calls, this single thread looked up the status of ports. If it did not find a port waiting for an incoming call, it would then record a switched call "loss"; otherwise, it would notify the Master Logic Support to bind the switched call to an RPC client that is awaiting a call (in this case, an incoming switched call "success" is recorded).

Detailed Description Text (55):
FIG. 5 illustrates the simulation of clients issuing outgoing calls. After starting the workstation 60, common variables and protocols were initiated 61. Multiple independent threads were issued 62 to simulated multiple user devices seeking to issue multiple outgoing calls through the gateway server. For each single simulated device started 63, the thread was put to sleep for a random period of time 64 to simulate the random access to the gateway's resources from a real LAN set-up.

Detailed Description Text (56):
After "waking" the device thread 65, a thread for sequential procedure calls for each thread simulating a single connection request was dispatched 66. A procedure PortOpen call was issued to the gateway server for each simulated client application thread 67.

Detailed Description Text (58):
A simulation of an incoming call is illustrated in FIG. 6. After starting this workstation 74, common variables and protocols were again assigned 75, and multiple independent child threads (corresponding to the number of simulated ports) to simulate LAN servers for remote clients on a switched network, dispatched 76, while the main thread remained in control forever 77.

Current US Cross Reference Classification (2):
718/102

First Hit Fwd Refs

Generate Collection

L9: Entry 7 of 10 File: USPT Nov 10, 1998

US-PAT-NO: 5835763
DOCUMENT-IDENTIFIER: US 5835763 A

TITLE: Threaded environment for computer systems without native threading support

DATE-ISSUED: November 10, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Klein; Paul F.	Thousand Oaks	CA		

US-CL-CURRENT: 718/101; 718/107

ABSTRACT:

A single threaded computer is provided with a threaded environment by using thread related batch jobs to accept thread tasks and individually apply them to specialized batch jobs that perform the tasks. Return data is applied by the thread related batch jobs to a common queue and the using application is caused to wait on the common queue, thereby overcoming the single threaded computer limitation that an application can only wait on one queue at a time. The thread tasks are thereby performed asynchronously while the user application effectively waits on all such thread tasks by waiting on the common queue. The threaded environment maintains a pool of active thread related batch jobs and permits the cloning of new thread related batch jobs from existing thread related batch jobs to efficiently manage the thread pool.

12 Claims, 2 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 1